A Realizability Interpretation for Intersection and Union Types

Daniel J. Dougherty, Ugo de 'Liguoro, Luigi Liquori, Claude Stolze

A Realizability Interpretation for Intersection and Union Types*

Daniel J. Dougherty[1], Ugo de'Liguoro[2], Luigi Liquori[3], and Claude Stolze[4]

[1] Worcester Polytechnic Institute, USA
dd@cs.wpi.edu
[2] Università di Torino, Italy
ugo.deliguoro@unito.it
[3] INRIA Sophia Antipolis-Méditerranée, France
Luigi.Liquori@inria.fr
[4] ENS Rennes and UPMC, France
Claude.Stolze@ens-rennes.fr

Abstract. *Proof-functional* logical connectives allow reasoning about the structure of logical proofs, in this way giving to the latter the status of *first-class* objects. This is in contrast to classical *truth-functional* connectives where the meaning of a compound formula is dependent only on the truth value of its subformulas.
In this paper we present a typed lambda calculus, enriched with strong products, strong sums, and a related proof-functional logic. This calculus, directly derived from a typed calculus previously defined by two of the current authors, has been proved isomorphic to the well-known Barbanera-Dezani-Ciancaglini-de'Liguoro type assignment system. We present a logic $\mathcal{L}^{\cap\cup}$ featuring two proof-functional connectives, namely strong conjunction and strong disjunction. We prove the typed calculus to be isomorphic to the logic $\mathcal{L}^{\cap\cup}$ and we give a realizability semantics using *Mints' realizers* [Min89] and a completeness theorem. A prototype implementation is also described.

1 Introduction

This paper is a contribution to the study of intersection and union type systems and their role in *logical* investigations.

There are two well-known points of view on type systems: (i) types as specifications and terms as programs, and (ii) types as propositions and terms as evidence. Let us call the former the "computational" perspective, and the latter the "logical" one.

In the logical view a type judgment $t : \sigma$ is taken to mean that t is a construction providing evidence of the proposition σ, reducing to a canonical element of σ. Typed λ-calculi defined in this way are at the core of proof assistants and logical frameworks. On the other hand, in the computational view a judgment

* Work supported by the COST Action CA15123 EUTYPES "The European research network on types for programming and verification".

$t : \sigma$ is taken to mean that t denotes an element of the datatype σ, which may in fact be defined in a way external to the system for making type-judgments.

Within the computational tradition itself there are two approaches: explicitly-typed calculi ("Church-style") and type assignment systems ("Curry-style"). These represent more than a difference in presentation: in type assignment systems types provide a means for making assertions about the semantics of raw terms, while in explicitly typed calculi types are a method of insuring that only well-behaved terms are considered at all.

The logical view resides naturally in a system of Church-style explicit typing. Existing logical frameworks and proof assistants take such explicitly-typed calculi for their foundation.

Intersection types originated within the computational perspective as a tool for analyzing the functional behavior of λ-terms: intersection type systems give characterizations of each of the sets of strongly normalizing, weakly normalizing, and head-normalizing terms [Pot80,CDC80,BCDC83]. From a programming languages perspective, intersection types support (finitary) *overloading*. Subtyping arises naturally in the study of intersection types.

Later, union types were introduced, as a foundational study [BDCd95] and also from programming languages motivation [MPS86,CF93,Dun12]. Union types are somewhat similar to sum types, but as Pierce [Pie02] notes: *"The main formal difference between disjoint and non-disjoint union types is that the latter lack any kind of* case *construct: if we know only that a value* v *has type* $T_1 \cup T_2$ *then the only operations we can safely perform on* v *are ones that make sense for* both T_1 *and* T_2*".*

Naturally, the question arose whether intersection, union, and subtyping can be given a logical explanation. Pottinger [Pot80] already identified this question: *"Since the meaning of* $\cap$ *is reasonably clear (to claim that* $A \cap B$ *is to claim that one has a reason for asserting* A *which is also a reason for asserting* B*), it would obviously be of interest to figure out how to add* $\cap$ *to intuitionist logic and then consider the analysis of intuitionist mathematical reasoning in the light of the resulting system"*. A natural logical analogue of computational interpretation of union types is "if we want to reason from an assumption v that $T_1 \cup T_2$ holds, then we may reason separately assuming v is evidence of T_1 and that v is evidence of T_2 as long as we use that evidence *in the same way.*"

There has subsequently been a lot of work on this question of understanding "proof-functional" connectives [MR72,LE85,Min89,AB91,BM94,DCGV97] where the logical analogue of intersection has come to be called "strong conjunction", with "strong disjunction" corresponding to union of course, and, in [DCGV97] with subtyping associated with "relevant implication", long of interest to philosophers. It became clear that a focus on *realizability* was most fruitful, typically taking untyped terms (from λ-calculus or combinatory logic) as realizers.

Independent of this thread of research, the question arose whether intersection and union type systems could be presented naturally in Church-style, *i.e.* explicitly typed. There are technical obstacles to an explicitly-typed treatment that would inherit the core properties of the type-assignment approach: subject reduc-

tion, subject expansion, strong normalization, unicity of typing, decidability of type reconstruction and type checking. Several proposals [PT94,Rey96,CLV01] [Ron02,WDMT02,WH02,Dun12] were explored, none of which met all the criteria above. The system presented here derives from the system of $\Lambda_{\mathsf{t}}^{\cap}$ [LR07] subsequently generalized in the system $\Lambda_{\mathsf{t}}^{\cap\cup}$ [DL10] to include union types. These systems do satisfy the core properties listed above. They do not include subtyping, and left open the question of a logical interpretation of the λ-calculus presented.

All of the work on understanding the logical aspects of intersection, union, and subtyping took place in the Curry-style framework. This was natural given the fact that type assignment was the most natural framework for intersection and union types, because the typing rules are not *syntax directed.* But the fact that most uses of λ-calculi in logical systems use explicitly-typed terms poses a compelling question, the main topic of the current paper:

> *Can a logical investigation of intersection and union types, with/without subtyping, take place in the context of an explicitly-typed λ-calculus?*

The motivation is that success here should point the way towards applications of intersection and union types in proof assistants and logical frameworks. The hope is that they can provide as much insight into logical systems as they have in the computational arena.

1.1 Contributions.

Our results can be thought of as exploring the relationships between the following four formal systems:

- the original system $\Lambda_{\mathsf{u}}^{\cap\cup}$ for type assignment with intersection and union types from [BDCd95],
- the typed calculus $\Lambda_{\mathsf{t}}^{\cap\cup}$ for type assignment with intersection and union types defined in [DL10],
- the proof-functional logic $\mathcal{L}^{\cap\cup}$, defined in this paper, and
- a natural deduction system $\mathsf{NJ}(\beta)$ for derivations in first-order intuitionistic logic with untyped λ-terms.

Judgements in these systems take the following four forms below. On the right-hand sides of the turnstiles, M is an untyped λ-term, Δ is a simply-typed λ-term with strong products and strong sums, and σ is a simple type formed using $\rightarrow, \cap,$ and $\cup$. The $r_\sigma[M]$ are typing predicates to be realized.

$$
\begin{array}{lllllll}
\mathbf{\Lambda}_{\mathsf{u}}^{\cap\cup} & B, & x_\iota & : \tau & \vdash & M & : \sigma \\
\mathbf{\Lambda}_{\mathsf{t}}^{\cap\cup} & \Gamma^{@}, & x_\iota \mathbin{@} \iota & : \tau & \vdash & M \mathbin{@} \Delta & : \sigma \\
\mathcal{L}^{\cap\cup} & \Gamma, & \iota & : \tau & \vdash & \Delta & : \sigma \\
\mathsf{NJ}(\beta) & G, & r_\tau[x_\iota] & & \vdash & r_\sigma[M] &
\end{array}
$$

The relationship between $\Lambda_{\mathsf{t}}^{\cap\cup}$ and $\Lambda_{\mathsf{u}}^{\cap\cup}$ was explored in [DL10], and is recalled in Section 2. The first contribution of this paper is the definition of a new notion, the *essence* $\wr\Delta\wr$ of a typed term Δ, used to connect $\Lambda_{\mathsf{t}}^{\cap\cup}$ and $\mathcal{L}^{\cap\cup}$. Specifically, we prove, as Theorem 6,

$$\Gamma^{@} \vdash M@\Delta : \sigma \text{ if and only if } \Gamma \vdash \Delta : \sigma \text{ and } \wr\Delta\wr \sqsubseteq M. \tag{1}$$

Here Γ is obtained from $\Gamma^{@}$ by erasing all the "$x@$", and $\sqsubseteq$ is a suitable syntactic preorder on untyped λ-terms. This justifies thinking of $\mathcal{L}^{\cap\cup}$ as a proof-functional logic. We think of the $\Lambda_{\mathsf{t}}^{\cap\cup}$ as a bridge between the intersection and union type assignment system and the logic $\mathcal{L}^{\cap\cup}$.

Our second contribution is to show how $\Lambda_{\mathsf{t}}^{\cap\cup}$ supports a *realizability* analysis of $\mathcal{L}^{\cap\cup}$. In particular, Section 3 shows that

$$\Gamma^{@} \vdash M@\Delta : \sigma \text{ and only if } \Delta \text{ realizes } G_\Gamma \vdash r_\sigma[M]. \tag{2}$$

Together with the equivalence in (1) this represents a complete analysis of the relationship between Curry-style and Church-style typing and the associated logic for intersection and union.

Section 4 presents further theoretical and pragmatic developments. Subsection 4.1 extends the typed system and the logic by adding a natural notion of subtyping. This is represented in the type assignment system as a non-syntax-directed substitution rule, in the typed calculus as an explicit coercion, and in the logic calculus as another well-known proof-functional connective called *relevant implication*. In Subsection 4.2 we briefly describe our prototype implementation of the type checking and proof inhabitation for the system with intersection/strong conjunction and union/strong disjunction and coercions as relevant implication.

1.2 Related Work

There are far too many studies of type systems featuring intersection, union, and subtyping to identify individually here. We have tried to outline the main currents of research in the introduction; here we will mention some work that is directly related to the contributions of this paper.

The formal investigation of soundness and completeness for a notion of realizability was initiated by Lopez-Escobar [LE85] and subsequently refined by Mints [Min89]. It is Mints' approach that we build on here.

The connection between intersection types and relevant implication was noticed by Alessi and Barbanera in [AB91]. Barbanera and Martini [BM94] studied three proof-functional operators, namely the *strong conjunction*, the *relevant implication* (see Meyer-Routley's [MR72] system B^+), and the *strong equivalence* connective for double implication, relating those connectives with suitable type assignments system, a realizability semantics and a completeness theorem.

Dezani-Ciancaglini, Ghilezan, and Venneri [DCGV97], investigated a *Curry-Howard* interpretation of intersection and union types (for Combinatory Logic).

Let $B \triangleq \{x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n\}$ ($i \neq j$ implies $x_i \not\equiv x_j$), and $B, x{:}\sigma \triangleq B \cup \{x{:}\sigma\}$

$$\frac{}{B \vdash M : \omega}\ (\omega) \qquad \frac{x{:}\sigma \in B}{B \vdash x : \sigma}\ (Var)$$

$$\frac{B, x{:}\sigma_1 \vdash M : \sigma_2}{B \vdash \lambda x.M : \sigma_1 \to \sigma_2}\ (\to I) \qquad \frac{B \vdash M : \sigma_1 \to \sigma_2 \quad B \vdash N : \sigma_1}{B \vdash M\,N : \sigma_2}\ (\to E)$$

$$\frac{B \vdash M : \sigma_1 \quad B \vdash M : \sigma_2}{B \vdash M : \sigma_1 \cap \sigma_2}\ (\cap I) \qquad \frac{B \vdash M : \sigma_1 \cap \sigma_2 \quad i = 1,2}{B \vdash M : \sigma_i}\ (\cap E_i)$$

$$\frac{B \vdash M : \sigma_i \quad i = 1,2}{B \vdash M : \sigma_1 \cup \sigma_2}\ (\cup I_i) \qquad \frac{\begin{array}{l} B, x{:}\sigma_1 \vdash M : \sigma_3 \\ B, x{:}\sigma_2 \vdash M : \sigma_3 \quad B \vdash N : \sigma_1 \cup \sigma_2 \end{array}}{B \vdash M[N/x] : \sigma_3}\ (\cup E)$$

Fig. 1. The Intersection and Union Type Assignment System $\Lambda^{\cap\cup}_{\mathsf{u}}$ [BDCd95].

Using the well understood relation between *combinatory logic* and λ-calculus, they encode type-free λ-terms in suitable combinatoric logic formulas and then type them using intersection and union types. As they put it, their goal is *"... to set out a logical system ... such that the intersection and union type constructors are interpreted as propositional connectives and then their derivability is completely represented by derivability in a logical Hilbert-style, axiomatization."* This is a complementary approach to the realizability-based one here.

Barbanera, Dezani-Ciancaglini, and de'Liguoro [BDCd95] presented an untyped λ-calculus with related type assignment system featuring intersection and union types. The previous work [DL10] presented a typed calculus that explored the relationship between the proof-functional intersections and unions and the truth-functional (strong) products and (strong) sums; the intersection and union aspect of the system was isomorphic, after erasure, to the Barbanera-Dezani-Ciancaglini-de'Liguoro [BDCd95] type assignment system. The type system we consider is built out of an infinitely enumerable set of type variables $\phi_0, \phi_1, \ldots$ and the constant type ω, by means of the arrow ("$\to$"), union ("$\cup$"), and intersection ("$\cap$") constructors. Therefore, types have the following syntax:

$$\sigma ::= \phi \mid \omega \mid \sigma \to \sigma \mid \sigma \cup \sigma \mid \sigma \cap \sigma.$$

2 Type Assignment $\Lambda^{\cap\cup}_{\mathsf{u}}$ and the Typed Calculus $\Lambda^{\cap\cup}_{\mathsf{t}}$

The type assignment system $\Lambda^{\cap\cup}_{\mathsf{u}}$ is the set of inference rules for assigning intersection and union types to terms of the pure λ-calculus. The presentation here, in Figure 1, is taken from [BDCd95]: the terms are standard raw λ-terms, and the types are generated from a set of base types by the constructors $\to, \cap$, and $\cup$.

$\Gamma^{@} \triangleq \{x_{\iota_1}@\iota_1{:}\sigma_1, \ldots, x_{\iota_n}@\iota_n{:}\sigma_n\}$, where $\iota_i \neq \iota_j$ implies $x_{\iota_i} \neq x_{\iota_j}$, and
$\Gamma^{@}, x_\iota@\iota{:}\sigma \triangleq \Gamma^{@} \cup \{x_\iota@\iota{:}\sigma\}$

$$\frac{}{\Gamma^{@} \vdash M@* : \omega}\ (\omega) \qquad \frac{x_\iota@\iota{:}\sigma \in \Gamma^{@}}{\Gamma^{@} \vdash x_\iota@\iota : \sigma}\ (Var)$$

$$\frac{\Gamma^{@}, x_\iota@\iota{:}\sigma_1 \vdash M@\Delta : \sigma_2}{\Gamma^{@} \vdash \lambda x_\iota.M@\lambda\iota{:}\sigma_1.\Delta : \sigma_1 \to \sigma_2}\ (\to I) \qquad \frac{\Gamma^{@} \vdash M@\Delta_1 : \sigma_1 \to \sigma_2 \quad \Gamma^{@} \vdash N@\Delta_2 : \sigma_1}{\Gamma^{@} \vdash M\,N@\Delta_1\,\Delta_2 : \sigma_2}\ (\to E)$$

$$\frac{\Gamma^{@} \vdash M@\Delta_1 : \sigma_1 \quad \Gamma^{@} \vdash M@\Delta_2 : \sigma_2}{\Gamma^{@} \vdash M@\langle \Delta_1\,,\, \Delta_2\rangle : \sigma_1 \cap \sigma_2}\ (\cap I) \qquad \frac{\Gamma^{@} \vdash M@\Delta : \sigma_1 \cap \sigma_2 \quad i \in \{1,2\}}{\Gamma^{@} \vdash M@\mathsf{pr_i}\Delta : \sigma_i}\ (\cap E_i)$$

$$\frac{\Gamma^{@} \vdash M@\Delta : \sigma_i \quad i \in \{1,2\}}{\Gamma^{@} \vdash M@\mathsf{in_i}\Delta : \sigma_1 \cup \sigma_2}\ (\cup I_i)$$

$$\frac{\Gamma^{@}, x_\iota@\iota{:}\sigma_1 \vdash M@\Delta_1 : \sigma_3 \quad \Gamma^{@}, x_\iota@\iota{:}\sigma_2 \vdash M@\Delta_2 : \sigma_3 \quad \Gamma^{@} \vdash N@\Delta_3 : \sigma_1 \cup \sigma_2}{\Gamma^{@} \vdash M\{N/x_\iota\}@[\overline{\lambda}\iota{:}\sigma_1.\Delta_1\,,\, \overline{\lambda}\iota{:}\sigma_2.\Delta_2] \cdot \Delta_3 : \sigma_3}\ (\cup E)$$

Fig. 2. The Typed Calculus $\Lambda_{\mathsf{t}}^{\cap\cup}$ [DL10].

Theorem 1 (Main properties of $\Lambda_{\mathsf{u}}^{\cap\cup}$ [BDCd95]).

Characterization. *The terms typable without use of the ω rule are precisely the strongly normalizing terms.* □

Parallel reduction. *If $B \vdash M : \sigma$ and $M \to_{gk} N$ then $B \vdash N : \sigma$. Here $\to_{gk}$ is the "Gross-Knuth" reduction, where all residuals of redexes in M are contracted (Def. 13.2.7 in [Bar84]).* □

In [DL10] a typed λ-calculus $\Lambda_{\mathsf{t}}^{\cap\cup}$ was defined, whose goal was to capture a decidable and Church-style version of the Curry-style $\Lambda_{\mathsf{u}}^{\cap\cup}$. The pseudo-terms of the $\Lambda_{\mathsf{t}}^{\cap\cup}$ calculus have the form $M@\Delta$, where M and Δ have the following syntax:

$$M ::= x_\iota \mid \lambda x_\iota.M \mid M\,M$$

$$\Delta ::= \iota \mid * \mid \lambda\iota{:}\sigma.\Delta \mid \Delta\,\Delta \mid \langle \Delta\,,\, \Delta\rangle \mid [\overline{\lambda}\iota{:}\sigma.\Delta\,,\, \overline{\lambda}\iota{:}\sigma.\Delta] \cdot \Delta \mid \mathsf{pr_i}\Delta \mid \mathsf{in_i}\Delta \quad i = 1,2$$

Note that the metasymbols $\overline{\lambda}$ and $\cdot$ are *per se* nothing but parts of the strong sum construction. The typed judgments are of the shape $\Gamma^{@} \vdash M@\Delta : \sigma$, where in a nutshell M is a type-free λ-term, Δ is a typed λ-term enriched with strong product, strong sum, projections, and injections to faithfully "memorize" every step of a type assignment derivation, and $\Gamma^{@}$ contains declarations of the shape

$x_\iota @\iota{:}\sigma$, where x_ι and ι are free-variables of M and Δ, respectively. The inference rules are presented in Figure 2. The main feature of the system was to keep M to be "synchronized" with Δ. As an example, we can derive the judgement $\vdash \lambda x_\iota.x_\iota @\langle \lambda\iota{:}\sigma_1.\iota\,,\, \lambda\iota{:}\sigma_2.\iota\rangle : (\sigma_1 \to \sigma_1) \cap (\sigma_2 \to \sigma_2)$. As another example, the term $[\overline{\lambda}\iota_1{:}\sigma_1.\Delta_1\,,\, \overline{\lambda}\iota_2{:}\sigma_2.\Delta_1]\cdot\Delta_3$ corresponds to the familiar case statement. The type ω plays the role of a terminal object, that is to say it is an object with a single element. The connection with type-assignment is this: every term can be assigned type ω so all proofs of that judgment have no content: all these proofs are considered identical ([Rey98], page 372). As is typical we name the unique element of the terminal object as $*$.

The relation between untyped and typed reductions is subtle because of the presence of the "Gross-Knuth" parallel reduction in the untyped calculus and a fairly complex notion of synchronization of M and Δ, via synchronized β- and Δ-reductions in the typed calculus. In a nutshell, for a given term $M@\Delta$, the computational part (M) and the logical part (Δ) grow up together while they are built through application of rules (Var), ($\to I$), and ($\to E$), but they *get disconnected* when we apply the ($\cap I$), ($\cup I$) or ($\cap E$) rules, which change the Δ but not the M. This disconnection is "logged" in the Δ via occurrences of $\langle -\,,\,-\rangle$, $[-\,,\,-]$, $\mathsf{pr_i}$, and $\mathsf{in_i}$. In order to correctly identify the reductions that need to be performed in parallel in order to preserve the correct syntax of the term, an *ad hoc* notion of *"overlapping"* that helps to define a redex taking into account the surrounding context was defined in [DL10]. Therefore, we define $\Rightarrow$ as the union of two reductions: $\Rightarrow_\beta$ dealing with β-reduction occurring in both M and Δ, and $\Rightarrow_\Delta$ dealing with reductions arising from reduction only in Δ. We refer to the complete reduction definition in [DL10]. Here are some main properties of the system $\Lambda_{\mathsf{t}}^{\cap\cup}$. Since the system is explicitly typed, properties such as type checking and type reconstruction are immediate.

Theorem 2 (Main properties of $\Lambda_{\mathsf{t}}^{\cap\cup}$ [DL10]).

Subject reduction. *If* $\Gamma^{@} \vdash M@\Delta : \sigma$ *and* $M@\Delta \Rightarrow M'@\Delta'$, *then* $\Gamma^{@} \vdash M'@\Delta' : \sigma$. □

Church-Rosser. *The reduction relation* $\Rightarrow$ *is confluent.* □

Strong normalization. *If* $M@\Delta$ *is typable without using rule* (ω) *then* M *is strongly normalizing.* □

Type reconstruction algorithm. *There is an algorithm* Type *satisfying*

Soundness. *If* $\mathsf{Type}(\Gamma^{@}, M@\Delta) = \sigma$, *then* $\Gamma^{@} \vdash M@\Delta : \sigma$. □

Completeness. *If* $\Gamma^{@} \vdash M@\Delta : \sigma$, *then* $\mathsf{Type}(\Gamma^{@}, M@\Delta) = \sigma$. □

Type checking algorithm. *There is an algorithm* $\mathsf{Typecheck}$ *satisfying* $\Gamma^{@} \vdash M@\Delta : \sigma$ *if and only if* $\mathsf{Typecheck}(\Gamma^{@}, M@\Delta, \sigma) = \mathsf{true}$.

Judgment decidability. *It is decidable whether* $\Gamma^{@} \vdash M@\Delta : \sigma$ *is derivable.* □

Isomorphism of typed-untyped derivations. *Let* $\mathcal{D}er\Lambda_{\mathsf{u}}^{\cap\cup}$ *and* $\mathcal{D}er\Lambda_{\mathsf{t}}^{\cap\cup}$ *be the sets of all (un)typed derivations. There are functions* $\mathcal{F} : \mathcal{D}er\Lambda_{\mathsf{t}}^{\cap\cup} \Rightarrow \mathcal{D}er\Lambda_{\mathsf{u}}^{\cap\cup}$ *and* $\mathcal{G} : \mathcal{D}er\Lambda_{\mathsf{u}}^{\cap\cup} \Rightarrow \mathcal{D}er\Lambda_{\mathsf{t}}^{\cap\cup}$ *showing the systems* $\Lambda_{\mathsf{t}}^{\cap\cup}$ *and* $\Lambda_{\mathsf{u}}^{\cap\cup}$ *to be isomorphic in the following sense:* $\mathcal{F} \circ \mathcal{G}$ *is the identity in* $\mathcal{D}er\Lambda_{\mathsf{u}}^{\cap\cup}$ *and* $\mathcal{G} \circ \mathcal{F}$ *is the identity in* $\mathcal{D}er\Lambda_{\mathsf{t}}^{\cap\cup}$ *modulo uniform naming of variable-marks,*

i.e., $\mathcal{G}(\mathcal{F}(\Gamma^{@} \vdash M@\Delta : \sigma)) = \mathsf{ren}(\Gamma^{@}) \vdash \mathsf{ren}(M@\Delta) : \sigma$*, where* ren *is a simple function renaming the free occurrences of variable-marks.* □

The algorithms Type and Typecheck in Figure 3 are exactly the ones from [DL10].

$$
\begin{array}{lcll}
\mathsf{Type}(\Gamma^{@}, M@\Delta) & \triangleq & \mathsf{match}\ M@\Delta\ \mathsf{with} & \\
\_@* & \Rightarrow & \omega & \\
\_@\mathsf{pr}_i\Delta_1 & \Rightarrow & \sigma_i \quad i=1,2 & \mathsf{if}\ \mathsf{Type}(\Gamma^{@}, M@\Delta_1) = \sigma_1 \cap \sigma_2 \\
\_@\langle \Delta_1\ ,\ \Delta_2 \rangle & \Rightarrow & \sigma_1 \cap \sigma_2 & \mathsf{if} \quad \mathsf{Type}(\Gamma^{@}, M@\Delta_1) = \sigma_1 \\
 & & & \mathsf{and}\ \mathsf{Type}(\Gamma^{@}, M@\Delta_2) = \sigma_2 \\
\_@\mathsf{in}_i\Delta_1 & \Rightarrow & \sigma_1 \cup \sigma_2 & \mathsf{if}\ \mathsf{Type}(\Gamma^{@}, M@\Delta_1) = \sigma_i \quad i=1,2 \\
\_@\left[\begin{array}{l}\overline{\lambda}\iota{:}\sigma_1.\Delta_1, \\ \overline{\lambda}\iota{:}\sigma_2.\Delta_2\end{array}\right]\cdot\Delta_3 & \Rightarrow & \sigma_3 & \mathsf{if} \quad \mathsf{Type}((\Gamma^{@}, x_\iota@\iota{:}\sigma_1), M'@\Delta_1) = \sigma_3 \\
 & & & \mathsf{and}\ \mathsf{Type}((\Gamma^{@}, x_\iota@\iota{:}\sigma_2), M'@\Delta_2) = \sigma_3 \\
 & & & \mathsf{and}\ \mathsf{Type}(\Gamma^{@}, N@\Delta_3) = \sigma_1 \cup \sigma_3\ \mathsf{and} \\
 & & & \mathsf{and}\ M \equiv M'[N/x] \\
x_\iota & \Rightarrow & \sigma & \mathsf{if}\ x_\iota@\iota{:}\sigma \in \Gamma^{@} \\
\lambda x_\iota.M_1@\lambda\iota{:}\sigma_1.\Delta_1 & \Rightarrow & \sigma_1 \to \sigma_2 & \mathsf{if}\ \mathsf{Type}((\Gamma^{@}, x_\iota@\iota{:}\sigma_1), M_1@\Delta_1) = \sigma_2 \\
M_1\ M_2@\Delta_1\ \Delta_2 & \Rightarrow & \sigma_2 & \mathsf{if} \quad \mathsf{Type}(\Gamma^{@}, M_1@\Delta_1) = \sigma_1 \to \sigma_2 \\
 & & & \mathsf{and}\ \mathsf{Type}(\Gamma^{@}, M_2@\Delta_2) = \sigma_1 \\
\_@\_ & \Rightarrow & \mathsf{false} & \mathsf{otherwise}
\end{array}
$$

$$\mathsf{Typecheck}(\Gamma^{@}, M@\Delta, \sigma) \triangleq \mathsf{Type}(\Gamma^{@}, M@\Delta) \stackrel{?}{=} \sigma$$

Fig. 3. The Type Reconstruction and Type Checking Algorithms for $\Lambda_{\mathsf{t}}^{\cap\cup}$.

2.1 The proof essence partial function

We start with a simple question: assuming $M@\Delta$ is derivable, can we *extract* the computational part M from a proof-term Δ? Luckily the answer is positive. To do that, let us extend the pure λ-calculus syntax by a constant Ω, typable by ω only, and consider the following pre-order join (partial) operation:

Definition 3. *Let* $\sqsubseteq$ *be the least pre-congruence over untyped* λ*-terms extended with the constant* Ω *such that:*

1. $\Omega \sqsubseteq M$ *for any* M
2. *if* $M =_\alpha M'$ *and* $M' \sqsubseteq N$ *then* $M \sqsubseteq N$
3. *if* $M =_\eta M'$ *and* $M' \sqsubseteq N$ *then* $M \sqsubseteq N$

By identifying η-convertible terms, the relation $\sqsubseteq$ is a partial order; next we show that the set of extended λ-terms is closed under join of compatible terms: M and N are compatible, written $M \uparrow N$, if $M \sqsubseteq P \sqsupseteq N$, for some P. Although the next lemma is intuitively clear its proof rather technical. We include the proof because existence of join of compatible terms is necessary for the subsequent definition of "essence" to make sense; it also provides a decision method for compatibility and a method to compute the join.

$$
\begin{array}{rll}
\Omega \vee M = M \vee \Omega = M & & \\
\lambda x.M \vee \lambda y.N = \lambda z.M[z/x] \vee N[z/y] & & z \text{ fresh} \\
M\,M' \vee N\,N' = (M \vee N)\,(M' \vee N') & & \\
M \vee N = \text{fail, else} & &
\end{array}
$$

Fig. 4. Syntactical join

Lemma 4. *For any pair M, N of extended λ-terms it is decidable whether they are compatible. Moreover, if $M \uparrow N$ then there exists a term $M \sqcup N$ which is the join of M and N w.r.t. $\sqsubseteq$ that is unique up to η-equality.*

Proof. First observe that if $M \sqsubseteq P =_\eta Q$ then $P \sqsubseteq Q$, as $\sqsubseteq$ includes $=_\eta$ and $M \sqsubseteq Q$, by transitivity of $\sqsubseteq$.

Let $\leq$ be the last pre-congruence such that $\Omega \leq M$, for any M. Then the relation $\sqsubseteq$ coincides with the transitive closure of $(=_\eta \leq) \cup (\leq =_\eta)$, where $M(=_\eta \leq)N$ if $M =_\eta P \leq N$ for some P, and similarly $M(\leq =_\eta)N$. Now suppose that

$$M =_\eta M' \leq P \geq N =_\eta N'$$

Since η-reduction is Church-Rosser and strongly normalizing, there exist the unique η-normal forms M'' of M, M', and P'' of P and N'' of N, N', respectively. By definition and the above remark we have $M'' \sqsubseteq P'' \sqsupseteq N''$; we claim that $M'' \leq P'' \geq N''$.

If $M' \leq P$, then for some context with n holes $C[\cdot]_1 \cdots [\cdot]_n$ we have $M' \equiv C[\Omega]_1 \cdots [\Omega]_n$ and $P \equiv C[P_1]_1 \cdots [P_n]_n$ for some P_i's. Assuming for simplicity $n = 1$ and that $M' \to_\eta M''$ in one step by contracting the η-redex $\lambda x.R\,x$, we either have that the hole filled by Ω does not occur in R or that R contains it. In the first case $\lambda x.R\,x$ is (the only) η-redex of P and we trivially obtain $P \to_\eta P'' \geq M''$ by contracting the same redex. In the second case Ω is a subterm of R which is such that $R \leq R'$ and $P \equiv C[\lambda x.R'\,x]$ for some R': then we have $M'' \equiv C[R] \leq C[R'] \equiv P''$ with $P \to_\eta P''$. The case of $n > 1$ or $M' \to_\eta^+ M''$ in several steps is a straightforward generalization thereof. By a similar reasoning we conclude that $P'' \geq N''$ as well. Also the proof that if

$$M \leq Q =_\eta Q' \geq N$$

Let $\Gamma \triangleq \{\iota_1{:}\sigma_1, \ldots, \iota_n{:}\sigma_n\}$, where $i \neq j$ implies $\iota_i \neq \iota_j$, and $\Gamma, \iota{:}\sigma \triangleq \Gamma \cup \{\iota{:}\sigma\}$

$$\frac{}{\Gamma \vdash * : \omega}\ (\omega) \qquad \frac{\iota{:}\sigma \in \Gamma}{\Gamma \vdash \iota : \sigma}\ (Var)$$

$$\frac{\Gamma, \iota{:}\sigma_1 \vdash \Delta : \sigma_2}{\Gamma \vdash \lambda\iota{:}\sigma_1.\Delta : \sigma_1 \to \sigma_2}\ (\to I) \qquad \frac{\Gamma \vdash \Delta_1 : \sigma_1 \to \sigma_2 \quad \Gamma \vdash \Delta_2 : \sigma_1}{\Gamma \vdash \Delta_1\, \Delta_2 : \sigma_2}\ (\to E)$$

$$\frac{\begin{array}{l}\Gamma \vdash \Delta_1 : \sigma_1 \\ \Gamma \vdash \Delta_2 : \sigma_2 \quad \wr\, \Delta_1 \,\wr \uparrow \wr\, \Delta_2 \,\wr\end{array}}{\Gamma \vdash \langle \Delta_1\, ,\, \Delta_2 \rangle : \sigma_1 \cap \sigma_2}\ (\cap I) \qquad \frac{\Gamma \vdash \Delta : \sigma_1 \cap \sigma_2 \quad i \in \{1,2\}}{\Gamma \vdash \mathsf{pr_i}\Delta : \sigma_i}\ (\cap E_i)$$

$$\frac{\Gamma \vdash \Delta : \sigma_i \quad i \in \{1,2\}}{\Gamma \vdash \mathsf{in_i}\Delta : \sigma_1 \cup \sigma_2}\ (\cup I_i) \qquad \frac{\begin{array}{l}\Gamma, \iota{:}\sigma_1 \vdash \Delta_1 : \sigma_3 \quad \wr\, \Delta_1 \,\wr \uparrow \wr\, \Delta_2 \,\wr \\ \Gamma, \iota{:}\sigma_2 \vdash \Delta_2 : \sigma_3 \quad \Gamma \vdash \Delta_3 : \sigma_1 \cup \sigma_2\end{array}}{\Gamma \vdash [\overline{\lambda}\iota{:}\sigma_1.\Delta_1\, ,\, \overline{\lambda}\iota{:}\sigma_2.\Delta_2] \cdot \Delta_3 : \sigma_3}\ (\cup E)$$

Fig. 5. The proof-functional logic $\mathcal{L}^{\cap\cup}$.

then $M'' \leq Q'' \geq N''$, where M'', Q'' and N'' are the respective η-normal forms of M, Q and Q', N is analogous.

From this it follows that if $M \sqsubseteq P \sqsupseteq N$, then $M'' \leq P'' \geq N''$ for their respective η-normal forms; as the inverse implication holds by definition, we can decide whether $M \uparrow N$ by reducing both M and N to their η-normal forms M'' and N'', and then deciding whether they are compatible w.r.t. the simpler relation $\leq$. In such a case we have that $M \sqcup N = M'' \vee N''$, where $\vee$ is defined in Figure 4, namely the lub w.r.t. $\leq$. □

Let us define the *essence* of a Δ, written $\wr\, \Delta \,\wr$, as a partial mapping as follows:

Definition 5 (Proof essence). *The type-free* essence M *of a typed proof* Δ *is:*

$$\begin{array}{rr}
\wr\, * \,\wr \triangleq \Omega & \wr\, \iota \,\wr \triangleq x_\iota \\
\wr\, \lambda\iota{:}\sigma_1.\Delta \,\wr \triangleq \lambda x_\iota.\wr \Delta \,\wr & \wr\, \Delta_1\, \Delta_2 \,\wr \triangleq \wr\, \Delta_1 \,\wr \wr\, \Delta_2 \,\wr \\
\wr\, [\overline{\lambda}\iota{:}\sigma_1.\Delta_1\, ,\, \overline{\lambda}\iota{:}\sigma_2.\Delta_2] \cdot \Delta_3 \,\wr \triangleq (\wr\, \Delta_1 \,\wr \sqcup \wr\, \Delta_2 \,\wr)\{\wr\, \Delta_3 \,\wr / x_\iota\} & \wr\, \mathsf{in_i}\Delta \,\wr \triangleq \wr\, \Delta \,\wr \\
\wr\, \langle \Delta_1\, ,\, \Delta_2 \rangle \,\wr \triangleq \wr\, \Delta_1 \,\wr \sqcup \wr\, \Delta_2 \,\wr & \wr\, \mathsf{pr_i}\Delta \,\wr \triangleq \wr\, \Delta \,\wr
\end{array}$$

The *"essence"* map is partial because join is such; it is however always defined when applied to a typed proof-term Δ in the typed calculus $\Lambda_{\mathsf{t}}^{\cap\cup}$ of [DL10] (see Theorem 6 below) and it produces a type-free λ-term M. Note that M and Δ are both typable with σ using the type assignment and the type system, respectively. Summarizing, the signature of the essence is as follows:

$$\wr\, - \,\wr : \text{proof-terms } (\Delta\text{'s}) \to \text{untyped } \lambda\text{-terms } (M\text{'s}).$$

2.2 The Proof-functional Logic $\mathcal{L}^{\cap\cup}$

Indeed, for a given typable Δ, the left-hand side of the @, namely M, can be omitted since it represents just the essence of Δ, *i.e.* $\wr\,\Delta\,\wr \sqsubseteq M$. Thus we can introduce the proof-functional logic, called $\mathcal{L}^{\cap\cup}$ and presented in Figure 5. The following theorem holds:

Theorem 6 (Equivalence). *Let Γ be obtained by $\Gamma^{@}$, simply by erasing all the "$x@$". Then $\Gamma^{@}\vdash M@\Delta : \sigma$ if and only if $\Gamma \vdash \Delta : \sigma$ and $\wr\,\Delta\,\wr \sqsubseteq M$.* □

Proof. The left-to-right is by induction over the the derivation of $\Gamma^{@}\vdash M@\Delta : \sigma$. First observe that if the derivation consists of axiom (ω) then $\Delta \equiv *$ and $\sigma = \omega$ and $\wr * \wr = \Omega \sqsubseteq M$. If the derivation ends by

$$\frac{\Gamma^{@}\vdash M@\Delta_1 : \sigma_1 \quad \Gamma^{@}\vdash M@\Delta_2 : \sigma_2}{\Gamma^{@}\vdash M@\langle \Delta_1\,,\,\Delta_2\rangle : \sigma_1 \cap \sigma_2}\ (\cap I)$$

then by induction we have that both $\wr\,\Delta_1\,\wr$ and $\wr\,\Delta_2\,\wr$ are defined and that $\wr\,\Delta_1\,\wr \sqsubseteq M \sqsupseteq \wr\,\Delta_2\,\wr$, therefore $\wr\,\langle \Delta_1\,,\,\Delta_2\rangle\,\wr = \wr\,\Delta_1\,\wr \sqcup \wr\,\Delta_2\,\wr$ is defined and $\wr\,\Delta_1\,\wr \sqcup \wr\,\Delta_2\,\wr \sqsubseteq M$ as desired.

If the derivation ends by $(\cup E)$ we reason in the same way as in case $(\cap I)$, while all other cases are immediate by induction and the fact that $\sqsubseteq$ is a pre-congruence.

The converse direction is is a straightforward induction over the derivation of $\Gamma \vdash \Delta : \sigma$. □

Since $\mathcal{L}^{\cap\cup}$ is a proof-functional logic it is natural to consider the pair "$\Delta : \sigma$" as a logical formula. Pictorially speaking, we could say that the type assignment system of [BDCd95] and the logic $\mathcal{L}^{\cap\cup}$ are "bridged" by the typed system $\Lambda_{\mathsf{t}}^{\cap\cup}$, and the above. We prove this fact by means of the concept of essence. This is, to the best of our knowledge, the first attempt to interpret union as a proof-functional connective.

3 Realizability interpretation of union types

In contrast to the system of intersection types, the type assignment system $\Lambda_{\mathsf{u}}^{\cap\cup}$ has no simple set-theoretic interpretation (see [BDCd95]). On the other hand system $\Lambda_{\mathsf{t}}^{\cap\cup}$ is grounded on the proof-functional logic $\mathcal{L}^{\cap\cup}$, though this is hardly standard. In this section we provide both a natural semantics for union types and a foundation for the logic $\mathcal{L}^{\cap\cup}$. We do this by interpreting the union type assignment system into the intuitionistic first order theory $\mathsf{NJ}(\beta)$, Mint's provable realizability of intersection types extended with union. Then we prove that the Δ's terms of system $\Lambda_{\mathsf{t}}^{\cap\cup}$ are just proof-terms in $\mathsf{NJ}(\beta)$.

From Theorem 6 we know that if $\Gamma^{@} \vdash M@\Delta : \sigma$, then there is a tight relation among Δ and M, which is captured by the essence mapping. Comparing system $\Lambda_{\mathsf{t}}^{\cap\cup}$ to the original $\Lambda_{\mathsf{u}}^{\cap\cup}$ it is easily seen that Δ is a proof-term of the

statement $M : \sigma$ in system $\Lambda^{\cap\cup}_{\mathsf{u}}$. But Δ is a simply typed term: in fact if we drop the restriction concerning the "essence" in rules $(\cap I)$ and $(\cup E)$ in system $\mathcal{L}^{\cap\cup}$ replacing $\sigma \cap \tau$ by $\sigma \times \tau$ and $\sigma \cup \tau$ by $\sigma + \tau$ then we get a simply typed λ-calculus with product and sums, namely the intuitionistic propositional logic with implication, conjunction, and disjunction in disguise.

We will provide a foundation for the proof-functional logic $\mathcal{L}^{\cap\cup}$ by interpreting the $\mathcal{L}^{\cap\cup}$ into an extension of Mints' provable realizability. However when proving a formula $r_\sigma[M]$ we have *two kinds* of realizers: the former is the untyped λ-term M, that we propose to call just a "method" borrowing terminology from Barbanera-Martini, the latter kind are Δ's that turn out to be realizers in the ordinary sense of intuitionistic logic.

Therefore, we prove a completeness proof that this is the case, namely that $\Gamma \vdash \Delta : \sigma$ is derivable in $\mathcal{L}^{\cap\cup}$ if and only if Δ realizes $G_\Gamma \vdash r_\sigma[M]$ for some M related to Δ by the essence mapping.

For this aim we use and extend Mints' approach of Provable Realizability [Min89,AB91,BM94]. We interpret the statement $\vdash M@\Delta : \sigma$ as "Δ is a construction of $M : \sigma$"; on the other hand $M : \sigma$ is the meaning of the formula $r_\sigma[M]$, provided that we extend the notion to cope with union types; the latter formula reads as "M is a method to assess σ" in terms of [LE85,BM94]; now the meaning of Δ is that of a constructive proof of $r_\sigma[M]$, and hence it is a "realizer" of this formula. In short we have "two kinds" of realizers on two levels: the M, which is a Mints' realizer of σ, and the Δ which is an ordinary realizer, in the sense of standard Brouwer–Heyting–Kolmogorov interpretation of intuitionistic logic, of the statement $r_\sigma[M]$.

To avoid confusion, in the following we shall reserve the word "realizer" for the Δ-terms, and we will use the word "method" referring to the untyped λ-term M.

Definition 7. *Let $\mathbf{P}_\phi(x)$ be a unary predicate for each atomic type ϕ. Then we define the predicates $r_\sigma[M]$ for types σ and terms M by induction over σ, as the first order logical formulae:*

$$\begin{aligned} r_\phi[x] &\equiv \mathbf{P}_\phi(x) \\ r_{\sigma_1\to\sigma_2}[x] &\equiv \forall y.r_{\sigma_1}[y] \supset r_{\sigma_2}[x\,y] \\ r_{\sigma_1\cap\sigma_2}[x] &\equiv r_{\sigma_1}[x] \wedge r_{\sigma_2}[x] \\ r_{\sigma_1\cup\sigma_2}[x] &\equiv r_{\sigma_1}[x] \vee r_{\sigma_2}[x] \end{aligned}$$

In the above $\supset$, $\wedge$ and $\vee$ are the logical connectives for implication, conjunction and disjunction respectively, that must be kept distinct from $\cap$ and $\cup$. In the first order language whose terms are type-free λ-terms, we have formulas of the shape $r_\sigma[M]$, whose intended meaning is that M is a method for σ in the intersection-union type discipline. Note that in $r_\sigma[x]$ the term-variable x is the only free-variable; in particular in $r_{\sigma_1\to\sigma_2}[M] \equiv \forall y.r_{\sigma_1}[y] \supset r_{\sigma_2}[M\,y]$ we assume that $y \notin \mathsf{Fv}(M)$.

By NJ we mean the natural-deduction presentation of the intuitionistic first-order predicate calculus. Derivations in NJ are trees of judgments $G \vdash A$, where

G is the set of undischarged assumptions, rather than trees of formulas as in Gentzen's original formulation.

Definition 8 (The system $\mathsf{NJ}(\beta)$). *The system $\mathsf{NJ}(\beta)$ is the natural deduction system for first order intuitionistic logic with untyped λ-terms and predicates $\mathbf{P}_\phi(x)$, the latter being axiomatized via the Post rules:*

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \mathbf{P}_\phi(M) \quad M =_{\beta\eta} N}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \mathbf{P}_\phi(N)}\ (Ax\beta\eta) \qquad \frac{}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} \mathbf{P}_\omega(M)}\ (Ax\omega)$$

If A is a formula of $\mathsf{NJ}(\beta)$ and $G \triangleq \{A_1, \ldots, A_n\}$ is a set of formulæ (a context), then we write $G \vdash_{\mathsf{NJ}(\beta)} A$ to mean that A is derivable in G. To the context $\Gamma \triangleq \{\iota_1{:}\sigma_1, \ldots, \iota_n{:}\sigma_n\}$ of the logic $\mathcal{L}^{\cap\cup}$ we associate the $\mathsf{NJ}(\beta)$ context $G_\Gamma \triangleq r_{\sigma_1}[x_{\iota_1}], \ldots, r_{\sigma_n}[x_{\iota_n}]$. Note that $G_{\Gamma,\iota:\sigma} \triangleq G_\Gamma, r_\sigma[x_\iota]$ and $x_\iota \notin \mathsf{Fv}(G_\Gamma)$, since $\iota \notin Dom(\Gamma)$, by context definition.

The following lemmas are useful to eliminate some of the intricacies of using derivations in the full system $\mathsf{NJ}(\beta)$, involving the universal quantifier in the definition of $r_{\sigma\to\tau}[M]$.

Lemma 9. *The following rule is admissible in $\mathsf{NJ}(\beta)$:*

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A\{M/x\} \quad M =_{\beta\eta} N}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A\{N/x\}}\ (Eq\beta\eta)$$

Proof. *By induction over the proof of $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} A\{M/x\}$.* □

Lemma 10. *The following rules are admissible in $\mathsf{NJ}(\beta)$:*

$$\frac{G_\Gamma, r_{\sigma_1}[x] \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_2}[M]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\to\sigma_2}[\lambda x.M]} \qquad \frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\to\sigma_2}[M] \quad G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1}[N]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_2}[M\,N]}$$

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1}[M] \quad G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_2}[M]}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\cap\sigma_2}[M]} \qquad \frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\cap\sigma_2}[M] \quad i \in \{1,2\}}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_i}[M]}$$

$$\frac{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_i}[M] \quad i \in \{1,2\}}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\cup\sigma_2}[M]} \qquad \frac{\begin{array}{l} G_\Gamma, r_{\sigma_1}[x] \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_3}[M] \\ G_\Gamma, r_{\sigma_2}[x] \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_3}[M] \quad G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_1\cup\sigma_2}[N] \end{array}}{G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_{\sigma_3}[M\{N/x\}]}$$

Proof. *In each case, use induction over the proof of the indicated premisses.* □

In spite of the similarity of the rules in Lemma 9 with those of system $\mathcal{L}^{\cap\cup}$ there are no restrictions on the shape of the derivations of the $r_\sigma[M]$. This is due to the fact the last lemma is about derivations of the predicate $r_\sigma[M]$ and not just of the proof-functional "formula" σ. Nonetheless we have:

Lemma 11. *If $\Gamma \overset{@}{\vdash} M@\Delta : \sigma$ in system $\Lambda^{\cap\cup}_{\mathsf{t}}$ then $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$.*

Proof. By induction over the derivation of $\Gamma^{@} \vdash M@\Delta : \sigma$ using Lemmas 9, and 10 □

Theorem 12 (Soundness). *If $\Gamma \vdash \Delta : \sigma$ is derivable in $\mathcal{L}^{\cap\cup}$ then there exists M such that $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$.*

Proof. By Theorem 6 if $\Gamma \vdash \Delta : \sigma$ is derivable then $\Gamma^{@} \vdash M@\Delta : \sigma$ for some $M \sqsupseteq \wr \Delta \wr$. The thesis follows by Lemma 11. □

We say that the derivation of $G_\Gamma \vdash r_\sigma[M]$ is *standard* if it uses only the rules of the Post system, rule $(Eq\beta\eta)$ and the rules from Lemmas 9 and 10; then we write $G_\Gamma \vdash_S r_\sigma[M]$.

Recall that $\mathsf{NJ}(\beta)$ is a particular case of systems called **I(S)** in [Pra71], which enjoys the property of being strongly normalizable. The normal form of a derivation, called "fully normal derivation" by Prawitz, is split into a topmost "analytical part" consisting of elimination rules, an intermediate "minimum part" consisting of rules of the Post system, and a final "synthetical part" (ending with the very conclusion of the derivation) only consisting of introduction rules. This implies the subformula property.

Lemma 13. *If $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$ then $G_\Gamma \vdash_S r_\sigma[M]$.*

Proof. By induction over the fully-normal derivation of $G_\Gamma \vdash r_\sigma[M]$, and then by cases of σ. If σ is ϕ or ω then both the analytic and the synthetic parts are empty, and the thesis is immediate. Otherwise:

Case $\sigma \equiv \sigma_1 \cap \sigma_2$. Since $r_{\sigma_1\cap\sigma_2}[M] \equiv r_{\sigma_1}[M] \wedge r_{\sigma_2}[M]$, the fully-normal derivation of $G_\Gamma \vdash r_{\sigma_1}[M] \wedge r_{\sigma_2}[M]$ must end with $(\wedge I)$, whose premises are $G_\Gamma \vdash r_{\sigma_i}[M]$, $i \in \{1,2\}$ and the thesis follows by induction.

Case $\sigma = \sigma_1 \to \sigma_2$. We have $r_{\sigma_1\to\sigma_2}[M] \equiv \forall y.r_{\sigma_1}[y] \supset r_{\sigma_2}[M\,y]$, so that the synthetic part ends by:

$$\dfrac{\dfrac{G_\Gamma, r_{\sigma_1}[y] \vdash r_{\sigma_2}[M\,y]}{G_\Gamma \vdash r_{\sigma_1}[y] \supset r_{\sigma_2}[M\,y]}\,(\supset I)}{G_\Gamma \vdash \forall y.r_{\sigma_1}[y] \supset r_{\sigma_2}[M\,y]}\,(\forall I)$$

where $y \notin \mathsf{Fv}(G_\Gamma) \cup \mathsf{Fv}(M)$ because of the side condition of rule $(\forall I)$ and the definition of $r_{\sigma_1\to\sigma_2}[M]$. By induction $G_\Gamma, r_{\sigma_1}[y] \vdash_S r_{\sigma_2}[M\,y]$, from which we obtain the standard derivation:

$$\dfrac{\dfrac{G_\Gamma, r_{\sigma_1}[y] \vdash_S r_{\sigma_2}[M\,y]}{G_\Gamma \vdash_S r_{\sigma_1\to\sigma_2}[\lambda y.M\,y]} \qquad \lambda y.M\,y =_\eta M}{G_\Gamma \vdash_S r_{\sigma_1\to\sigma_2}[M]}$$

Case $\sigma = \sigma_1 \cup \sigma_2$. Then $r_{\sigma_1\cup\sigma_2}[M] \equiv r_{\sigma_1}[M] \vee r_{\sigma_2}[M]$ and the fully-normal derivation of $G_\Gamma \vdash r_{\sigma_1}[M] \vee r_{\sigma_2}[M]$ ends by $(\vee I)$, therefore by induction $G_\Gamma \vdash_S r_{\sigma_i}[M]$ with $i \in \{1,2\}$ and the thesis follows. □

Definition 14 (Δ-realizability). *We say that a closed* Δ realizes *the formula* $r_\sigma[M]$*, written* $\Delta \Vdash r_\sigma[M]$*, if* $\wr\,\Delta\,\wr \sqsubseteq M$ *and:*

$$\begin{array}{rcl}
\Delta \Vdash r_\phi[M] & & \textit{always} \\
\Delta \Vdash r_\omega[M] & \Leftrightarrow & \Delta \equiv * \\
\Delta \Vdash r_{\sigma\to\tau}[M] & \Leftrightarrow & \exists M' =_{\beta\eta} M.\, \forall \Delta', N.\ \Delta' \Vdash r_\sigma[N] \Rightarrow (\Delta\,\Delta') \Vdash r_\tau[M'N] \\
\Delta \Vdash r_{\sigma\cap\tau}[M] & \Leftrightarrow & \Delta \equiv \langle \Delta_1\,,\,\Delta_2\rangle \wedge \Delta_1 \Vdash r_\sigma[M] \wedge \Delta_2 \Vdash r_\tau[M] \\
\Delta \Vdash r_{\sigma\cup\tau}[M] & \Leftrightarrow & (\Delta \xrightarrow{*} \mathsf{in}_1\Delta_1 \wedge \Delta_1 \Vdash r_\sigma[M]) \vee (\Delta \xrightarrow{*} \mathsf{in}_2\Delta_2 \wedge \Delta_2 \Vdash r_\tau[M])
\end{array}$$

We then define $\Delta \Vdash G_\Gamma \vdash r_\sigma[M]$ *where* Δ *is a possibly open term such that* $\mathsf{Fv}(\Delta) = \{\iota_1, \ldots, \iota_k\} \subseteq \mathsf{Fv}(\Gamma)$*, if and only if for all closed* $\Delta_1, \ldots, \Delta_k$ *and terms* $N_1, \ldots, N_k$ *such that* $\Delta_i \Vdash r_{\Gamma(\iota_i)}[N_i]$ *for all* $i = 1, \ldots, k$ *it is the case that (writing* $x_i \equiv x_{\iota_i}$*):*

$$\Delta\{\Delta_1/\iota_1\}\cdots\{\Delta_k/\iota_k\} \Vdash r_\sigma[M\{N_1/x_1\}\cdots\{N_k/x_k\}].$$

Lemma 15. *If* $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$ *then there exists* Δ *such that* $\Delta \Vdash G_\Gamma \vdash r_\sigma[M]$*.*

Proof. By Lemma 13 we can argue by induction over the standard derivation of $G_\Gamma \vdash r_\sigma[M]$. If it ends by a Post rule, then the thesis is trivial. Suppose that it ends by the inference

$$\frac{G_\Gamma \vdash r_{\sigma_1}[M] \quad G_\Gamma \vdash r_{\sigma_2}[M]}{G_\Gamma \vdash r_{\sigma_1\cap\sigma_2}[M]}$$

Then by induction there are Δ_1, Δ_2 such that $\wr\,\Delta_i\,\wr \sqsubseteq M$ and $\Delta_i \Vdash G_\Gamma \vdash r_{\sigma_i}[M]$. Taking $\Delta \equiv \langle \Delta_1\,,\,\Delta_2\rangle$ we have that $\wr\,\Delta_1\,\wr \sqsubseteq M \sqsupseteq \wr\,\Delta_2\,\wr$ and $\wr\,\Delta\,\wr = \wr\,\Delta_1\,\wr \sqcup \wr\,\Delta_2\,\wr \sqsubseteq M$ hence $\Delta \Vdash G_\Gamma \vdash r_{\sigma_1\cap\sigma_2}[M]$. All other cases are similar. □

Lemma 16. *If* $\Delta \Vdash G_\Gamma \vdash r_\sigma[M]$ *then there exists* N *and* Δ' *such that* $M =_{\beta\eta} N$ *and* $\Gamma \overset{@}{\vdash} N@\Delta' : \sigma$*.*

Proof. By induction over σ. □

Theorem 17 (Completeness). *If* $G_\Gamma \vdash_{\mathsf{NJ}(\beta)} r_\sigma[M]$ *then there exists* $N =_{\beta\eta} M$ *and* Δ *such that* $\Gamma \overset{@}{\vdash} N@\Delta : \sigma$ *and therefore* $\Gamma \vdash \Delta : \sigma$*.*

Proof. By the hypothesis and Lemma 15 we know that there is a Δ' such that $\Delta' \Vdash G_\Gamma \vdash r_\sigma[M]$. By Lemma 16 this implies that $\Gamma \overset{@}{\vdash} N@\Delta : \sigma$ for some Δ and $N =_{\beta\eta} M$, and we conclude by Theorem 6. □

4 Further logical developments and implementation

There is active ongoing work on both the theoretical and practical directions of this project.

4.1 Implicit subtyping as explicit coercions

The logic $\mathcal{L}^{\cap\cup}$ does not encompass the subtyping relation treated in [BDCd95], which extends the subtyping relation among intersection types introduced in [BCDC83]. Given such a relation $\leq$, the subsumption rule takes the form:

$$\frac{B \vdash M : \sigma \quad \sigma \leq \tau}{B \vdash M : \tau} \ (Sub)$$

This rule has a character similar to the intersection and union introduction rules because the subject M of the conclusion is the same as in the premise. This calls for a consistent treatment on the side of the Δ's that are typed terms. In [DL10] it was hinted that the subtyping as coercion should be the proper approach, in the sense that whenever $\sigma \leq \tau$ there should exist a coercion λ-term $coe_{\sigma\leq\tau} : \sigma \to \tau$ such that the following rule is sound:

$$\frac{\Gamma \vdash \Delta : \sigma \quad \sigma \leq \tau}{\Gamma \vdash (coe_{\sigma\leq\tau}\, \Delta) : \tau} \ (coe)$$

According to the logic $\mathcal{L}^{\cap\cup}$ this rule is sound if $\wr\, coe_{\sigma\leq\tau}(\Delta) \wr \sqsubseteq M$, while according to the realizability interpretation this is the case if realizers of $r_\sigma[M]$ are sent to realizers of $r_\tau[M]$. We argue that this is the case by showing that, at least for the type theory Ξ from [BDCd95], we could establish the following:

Conjecture 18. If $\sigma \leq \tau \in \Xi$ then there exists a combinator $coe_{\sigma\leq\tau}$ such that $\vdash coe_{\sigma\leq\tau} : \sigma \to \tau$ is a theorem of $\mathcal{L}^{\cap\cup}$ and $\wr\, coe_{\sigma\leq\tau} \wr \sqsubseteq \lambda x.x$.

We end this subsection by observing that Conjecture 18 is in accordance with the logical interpretation of intersection types proposed in [BM94]. In fact from the logical point of view, subtyping of intersection (and union) types corresponds to inject concepts and rules proper to the *Minimal Relevant Logical* system B^+ introduced by Meyer-Routley in '72. As nicely explained in the Barbanera-Martini paper, the *relevant implication*, denoted by $\supset_r$ from the logic side and $\to_r$ from the type side, captures the behavior of the coercion function $coe_{\sigma\leq\tau}$ as follows:

> *"To assert $\sigma \to_r \tau$ (read also $\sigma \leq \tau$) is to assert that any proof-inhabitant of σ* is also *a proof-inhabitant of τ".*

Our system then meets the latter requirement because any coercion is "essentially" the identity.

4.2 Logical Frameworks

The results presented here are part of a larger project to build a small logical framework, *à la* the Edinburgh Logical Framework [HHP93], featuring proof-functional logical connectives like strong conjunction (intersection) and strong sum (union), and allowing reasoning about the structure of logical proofs, in this

way giving to the latter the status of *first-class objects*. We could also mention the high expressivity of *ad hoc* (intersection) polymorphism, since it allows to typecheck the untyped λ-term abstraction $\lambda x.x\,x$ (self-application), essence of a suitable Δ term, with the intersection type $(\sigma \cap (\sigma \to \sigma)) \to \sigma$. Other insights could come in studying `case` constructs typechecked with union types.

Another positive outcome of this research line would be the introduction of proof-functional types into existing interactive theorem provers such as Coq [Coq16] or Isabelle [Isa16], and dependently typed programming languages such as Agda [Agd16], Epigram [Epi16], or Idris [Idr16].

Finally, other advances in research line could come in studying other proof-functional logical connectives, like *relevant implication* (where the implication is established by an identity map) and *strong equivalence* (where the two directions of the equivalence are established by mutually inverse maps), the two being proof-functional interpretations of subtyping and provable type isomorphism, respectively.

4.3 Prototype Implementation

Our current implementation experiments with a small kernel for a logical framework featuring union and intersection types satisfying the *De Brujin Principle* saying *"Keep the framework as weak as possible (A plea for weaker frameworks")*.

The prototype is written in the functional language ML. Its *Read-Eval-Print-Loop* (REPL) can read a file containing some signatures, and process it using a lexer, then a parser. Then it can do the following actions:

- type-check the proof
- normalize the proof using strong reduction
- add some definitions in the global context
- perform a (human interactive) type inhabitation algorithm

We are putting our current efforts into make the REPL to consider proofs (Δ terms) as a genuine first-class objects.

We implemented the $\Lambda_{\mathsf{t}}^{\cap\cup}$ calculus and the proof-functional logic $\mathcal{L}^{\cap\cup}$ as presented here. We have added a wildcard type called "?" to deal with union introduction, and we added an unification algorithm to apply eliminations rule for implication and union types. The actual type system also features a first implementation of dependent-types *à la* LF: explicit coercions and strong equivalence are on the top of our implementation' todo list. The aim of the prototype is to check the expressiveness of the proof-functional nature of the logical engine in the sense that when the user must prove *e.g.* a strong conjunction formula $\sigma_1 \cap \sigma_2$ obtaining (mostly interactively) a witness Δ_1 for σ_1, the prototype can "squeeze" the essence M of Δ_1 to accelerate, and in some case automatize, the construction of a witness Δ_2 proof for the formula σ_2 having the same essence M of Δ_1. Existing proof assistants could get some benefit if extended with a proof-functional logic. We are also started an encoding of the proof-functional operators of intersection and union in Coq. The actual state of the prototype can be retrieved at `https://github.com/cstolze/Bull`.

Acknowledgment. *We are grateful to the anonymous reviewers for their useful remarks.*

References

[AB91] Fabio Alessi and Franco Barbanera. Strong conjunction and intersection types. In *MFCS*, pages 64–73, 1991.

[Agd16] The Agda Programming Language. `http://wiki.portal.chalmers.se/agda/pmwiki.php`, 2016. [Online; accessed 2-september-2016].

[Bar84] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, revised edition, 1984.

[BCDC83] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A Filter Lambda Model and the Completeness of Type Assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

[BDCd95] Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. Intersection and union types: syntax and semantics. *Inf. Comput.*, 119(2):202–230, 1995.

[BM94] Franco Barbanera and Simone Martini. Proof-functional connectives and realizability. *Archive for Mathematical Logic*, 33:189–211, 1994.

[CDC80] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

[CF93] Mario Coppo and Alberto Ferrari. Type inference, abstract interpretation and strictness analysis. *Theoretical Computer Science*, 121(1):113–143, 1993.

[CLV01] Beatrice Capitani, Michele Loreti, and Betti Venneri. Hyperformulae, Parallel Deductions and Intersection Types. *BOTH, Electr. Notes Theor. Comput. Sci.*, 50(2):180–198, 2001.

[Coq16] The Coq Proof Assistant. `https://coq.inria.fr/`, 2016. [Online; accessed 2-september-2016].

[DCGV97] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, and Betti Venneri. The "relevance" of intersection and union types. *Notre Dame Journal of Formal Logic*, 38(2):246–269, 1997.

[DL10] Daniel J. Dougherty and Luigi Liquori. Logic and computation in a lambda calculus with intersection and union types. In *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, pages 173–191, 2010.

[Dun12] Joshua Dunfield. Elaborating intersection and union types. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, pages 17–28. ACM, 2012.

[Epi16] The Epigram Programming Language. `https://code.google.com/archive/p/epigram/`, 2016. [Online; accessed 2-september-2016].

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, 1993.

[Idr16] The Idris Programming Language. `http://www.idris-lang.org/`, 2016. [Online; accessed 2-september-2016].

[Isa16] The Isabelle Proof Assistant. `https://isabelle.in.tum.de/`, 2016. [Online; accessed 2-september-2016].

[LE85] Edgar G. K. Lopez-Escobar. Proof functional connectives. In *Methods in Mathematical Logic*, volume 1130 of *Lecture Notes in Mathematics*, pages 208–221. Springer-Verlag, 1985.

[LR07] Luigi Liquori and Simona Ronchi Della Rocca. Intersection typed system *à la* Church. *Information and Computation*, 9(205):1371–1386, 2007.

[Min89] Grigori Mints. The completeness of provable realizability. *Notre Dame Journal of Formal Logic*, 30(3):420–441, 1989.

[MPS86] David MacQueen, Gordon Plotkin, and Ravi Sethi. An ideal model for recursive polymorphic types. *Information and Control*, 71:95–130, 1986.

[MR72] Robert K Meyer and Richard Routley. Algebraic analysis of entailment I. *Logique et Analyse*, 15:407–428, 1972.

[Pie02] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[Pot80] Garrel Pottinger. A type assignment for the strongly normalizable λ-terms. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, 1980.

[Pra71] Dag Prawitz. Ideas and results in proof theory. In *Proceedings of the Second Scandinavian Logic Symposium*. North-Holland, 1971.

[PT94] Benjamin C. Pierce and David N. Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, 1994.

[Rey96] John C. Reynolds. Design of the programming language Forsythe. In O'Hearn and Tennent, editors, *Algol-like Languages*. Birkhauser, 1996.

[Rey98] John C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.

[Ron02] Simona Ronchi Della Rocca. Intersection typed lambda-calculus. *Electr. Notes Theor. Comput. Sci.*, 70(1), 2002.

[WDMT02] Joe B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Program.*, 12(3):183–227, 2002.

[WH02] Joe B. Wells and Christian Haack. Branching types. In *ESOP*, volume 2305 of *Lecture Notes in Computer Science*, pages 115–132. Springer-Verlag, 2002.